Universidade Federal de São Carlos
Departamento de Computação
Prof. Dr. Renato Bueno

Projeto e Implementação de Banco de Dados

Trabalho 1

Grupo 6
Integrantes:

Allan Mansilha Cidreira, 760565
Amanda Peixoto Manso, 759847
João Victor Mendes Freire, 758943
Julia Cinel Chagas, 759314

Consultas:

Consulta 1

a) Crie a consulta em SQL

```
SELECT pjnome
FROM projeto,
     departamento,
     dept_localizacoes
WHERE projeto.dnum = departamento.dnumero AND
      departamento.dnumero = dept_localizacoes.dnumero AND
      dept_localizacoes.dlocalizacao = 'San Francisco'
```

b) Execute a consulta em SQL utilizando EXPLAIN ANALYSE

```
Nested Loop  (cost=4.66..26.97 rows=10 width=10) (actual time=0.168..0.449 rows=6
loops=1)
   ->  Hash Join  (cost=4.51..25.25 rows=10 width=18) (actual time=0.158..0.426 rows=6
loops=1)
         Hash Cond: (projeto.dnum = dept_localizacoes.dnumero)
         ->  Seq Scan on projeto  (cost=0.00..18.00 rows=1000 width=14) (actual
time=0.021..0.187 rows=1000 loops=1)
         ->  Hash  (cost=4.50..4.50 rows=1 width=4) (actual time=0.053..0.053 rows=1
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on dept_localizacoes  (cost=0.00..4.50 rows=1 width=4)
(actual time=0.010..0.034 rows=1 loops=1)
                     Filter: ((dlocalizacao)::text = 'San Francisco'::text)
                     Rows Removed by Filter: 199
   ->  Index Only Scan using departamento_pkey on departamento  (cost=0.14..0.17 rows=1
width=4) (actual time=0.002..0.002 rows=1 loops=6)
         Index Cond: (dnumero = projeto.dnum)
         Heap Fetches: 6
 Planning Time: 2.040 ms
 Execution Time: 0.544 ms
(14 rows)
```

c) Tente otimizar a consulta utilizando índices

```
CREATE INDEX dnum_index ON projetos(dnum);

Gerando a seguinte saída:

                      Table "public.projeto"
    Column    |         Type          | Collation | Nullable | Default
--------------+-----------------------+-----------+----------+---------
 pjnome       | character varying(30) |           |          |
 pnumero      | integer               |           | not null |
 plocalizacao | character varying(30) |           |          |
 dnum         | integer               |           |          |
Indexes:
    "projeto_pkey" PRIMARY KEY, btree (pnumero)
    "dnum_index" btree (dnum)
    "local_index" btree (plocalizacao)
Foreign-key constraints:
    "projeto_dnum_fkey" FOREIGN KEY (dnum) REFERENCES
departamento(dnumero)
Referenced by:
    TABLE "trabalha_em" CONSTRAINT "trabalha_em_pno_fkey" FOREIGN KEY
(pno) REFERENCES projeto(pnumero)
```

d) Execute a consulta em SQL utilizando EXPLAIN ANALYSE

```
Nested Loop  (cost=4.79..7.82 rows=10 width=10) (actual time=0.088..0.105 rows=6
loops=1)
   ->  Hash Join  (cost=4.51..6.79 rows=1 width=8) (actual time=0.056..0.068 rows=1
loops=1)
         Hash Cond: (departamento.dnumero = dept_localizacoes.dnumero)
         ->  Seq Scan on departamento  (cost=0.00..2.00 rows=100 width=4) (actual
time=0.009..0.019 rows=100 loops=1)
         ->  Hash  (cost=4.50..4.50 rows=1 width=4) (actual time=0.029..0.029 rows=1
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on dept_localizacoes  (cost=0.00..4.50 rows=1 width=4)
(actual time=0.011..0.026 rows=1 loops=1)
                     Filter: ((dlocalizacao)::text = 'San Francisco'::text)
                     Rows Removed by Filter: 199
   ->  Index Scan using dnum_index on projeto  (cost=0.28..0.93 rows=10 width=14)
(actual time=0.030..0.033 rows=6 loops=1)
         Index Cond: (dnum = departamento.dnumero)
 Planning Time: 1.177 ms
 Execution Time: 0.132 ms
(13 rows)
```

e) Comente/justifique

A consulta realizada, sem otimizações, teve um tempo de execução de 0.544ms. Analisando as tabelas que a consulta utiliza, percebe-se que a tabela `projeto` é a mais computacionalmente custosa, por ter um número maior de dados quando comparada às outras tabelas. Ademais, nota-se que foram removidas 199 linhas pelo filtro de cidade = São Francisco, sendo selecionadas apenas 14 linhas de informação.

Desta forma, o grupo considerou que as linhas em que a cidade do departamento era São Francisco eram pouco frequentes, o que explica a utilização de `Index Scan` na busca. Levando estes elementos em consideração, o grupo optou por criar índices na tabela `projeto`, na coluna `dnum`, de forma que a consulta precisasse de menos comparações entre a tabela `projeto` e a tabela `departamento`.

Com isso, a consulta após a adição dos índices teve um tempo de execução final de 0.132ms, uma melhora aproximada de 76%.

Consulta 2

a) Crie a consulta em SQL

```sql
SELECT ssn,
       pnome,
       salario,
       pjnome
FROM (empregado
      JOIN
      trabalha_em
      ON empregado.ssn=trabalha_em.essn
     )
     JOIN
     projeto
     ON trabalha_em.pno=projeto.pnumero
WHERE horas = 12;
```

b) Execute a consulta em SQL utilizando EXPLAIN ANALYSE

```
Gather  (cost=7592.08..12389.53 rows=24767 width=28) (actual time=248.413..326.972
rows=24926 loops=1)
   Workers Planned: 1
   Workers Launched: 1
   ->  Hash Join  (cost=6592.08..8912.83 rows=14569 width=28) (actual
time=243.580..315.051 rows=12463 loops=2)
         Hash Cond: (trabalha_em.pno = projeto.pnumero)
         ->  Parallel Hash Join  (cost=6561.58..8843.92 rows=14569 width=22) (actual
time=242.856..308.852 rows=12463 loops=2)
               Hash Cond: (empregado.ssn = trabalha_em.essn)
               ->  Parallel Seq Scan on empregado  (cost=0.00..1829.24 rows=58824
width=18) (actual time=0.166..42.942 rows=50000 loops=2)
               ->  Parallel Hash  (cost=6379.47..6379.47 rows=14569 width=8) (actual
time=242.481..242.481 rows=12463 loops=2)
                     Buckets: 32768  Batches: 1  Memory Usage: 1280kB
                     ->  Parallel Seq Scan on trabalha_em  (cost=0.00..6379.47
rows=14569 width=8) (actual time=0.124..232.819 rows=12463 loops=2)
                           Filter: (horas = '12'::numeric)
                           Rows Removed by Filter: 237537
         ->  Hash  (cost=18.00..18.00 rows=1000 width=14) (actual time=0.616..0.617
rows=1000 loops=2)
               Buckets: 1024  Batches: 1  Memory Usage: 56kB
               ->  Seq Scan on projeto  (cost=0.00..18.00 rows=1000 width=14) (actual
time=0.025..0.397 rows=1000 loops=2)
 Planning Time: 18.884 ms
 Execution Time: 329.532 ms
(18 rows)
```

c) Tente otimizar a consulta utilizando índices

```
CREATE INDEX horas_index ON trabalha_em(horas);

Gerando a seguinte saída:
               Table "public.trabalha_em"
 Column |     Type     | Collation | Nullable | Default
--------+--------------+-----------+----------+---------
 essn   | integer      |           | not null |
 pno    | integer      |           | not null |
 horas  | numeric(3,1) |           |          |
Indexes:
    "trabalha_em_pkey" PRIMARY KEY, btree (essn, pno)
    "horas_index" btree (horas)
Foreign-key constraints:
    "trabalha_em_essn_fkey" FOREIGN KEY (essn) REFERENCES empregado(ssn)
    "trabalha_em_pno_fkey" FOREIGN KEY (pno) REFERENCES projeto(pnumero)
```

d) Execute a consulta em SQL utilizando EXPLAIN ANALYSE

```
 Hash Join  (cost=3821.04..7249.28 rows=24767 width=28) (actual time=38.401..82.101
rows=24926 loops=1)
   Hash Cond: (trabalha_em.pno = projeto.pnumero)
   ->  Hash Join  (cost=3790.54..7153.49 rows=24767 width=22) (actual
time=37.477..75.248 rows=24926 loops=1)
         Hash Cond: (empregado.ssn = trabalha_em.essn)
         ->  Seq Scan on empregado  (cost=0.00..2241.00 rows=100000 width=18) (actual
time=0.261..17.430 rows=100000 loops=1)
         ->  Hash  (cost=3480.95..3480.95 rows=24767 width=8) (actual
time=37.083..37.086 rows=24926 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 1230kB
               ->  Bitmap Heap Scan on trabalha_em  (cost=468.37..3480.95 rows=24767
width=8) (actual time=6.023..31.613 rows=24926 loops=1)
                     Recheck Cond: (horas = '12'::numeric)
                     Heap Blocks: exact=2703
                     ->  Bitmap Index Scan on horas_index  (cost=0.00..462.18 rows=24767
width=0) (actual time=5.296..5.297 rows=24926 loops=1)
                           Index Cond: (horas = '12'::numeric)
   ->  Hash  (cost=18.00..18.00 rows=1000 width=14) (actual time=0.884..0.889 rows=1000
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 56kB
         ->  Seq Scan on projeto  (cost=0.00..18.00 rows=1000 width=14) (actual
time=0.011..0.613 rows=1000 loops=1)
 Planning Time: 17.009 ms
 Execution Time: 84.166 ms
(17 rows)
```

e) Comente/justifique

A consulta realizada precisava conter três tabelas, inclusive a maior de todas. Depois de realizar um `SELECT count(*) FROM <tabelas>`, descobrimos que a tabela `trabalha_em` era a maior (seguida de `dependentes` e `empregado`).

Foi criada uma consulta que selecionasse nome, salário, ssn e nome do projeto de empregados que trabalham 12 horas em algum projeto. Isso exigiria juntar as tabelas `empregado`, `projeto` e `trabalha_em`.

Na primeira execução o tempo da consulta foi de aproximadamente *330 ms*. Quando consultado novamente, o *cache* reduziu o tempo para aproximadamente *200 ms*. Com o objetivo de melhorar esse desempenho, foi criado um índice na coluna `horas` de `trabalha_em`. Essa otimização foi suficiente para reduzir a busca para aproximadamente *85ms*, uma redução de 75% do tempo original. Essa redução se deu uma vez que, ao invés de pesquisar sequencialmente em uma tabela com 500.000 linhas, o índice já mostrava as com o valor desejado, e portanto o tempo gasto foi realizando as junções da consulta.